JAVA

Interfaces Graphiques SWING

Introduction

- `import javax.swing.*;`
- Principales caractéristiques de SWING
- Une petite interface
- La gestion des évènements
- Les principaux composants SWING
- La disposition des composants

- Attention : ce cours ne montre que quelques unes des nombreuses possibilités offertes par les composants
  - http://java.sun.com/docs/books/tutorial/uiswing/components/

Principales caractéristiques de SWING

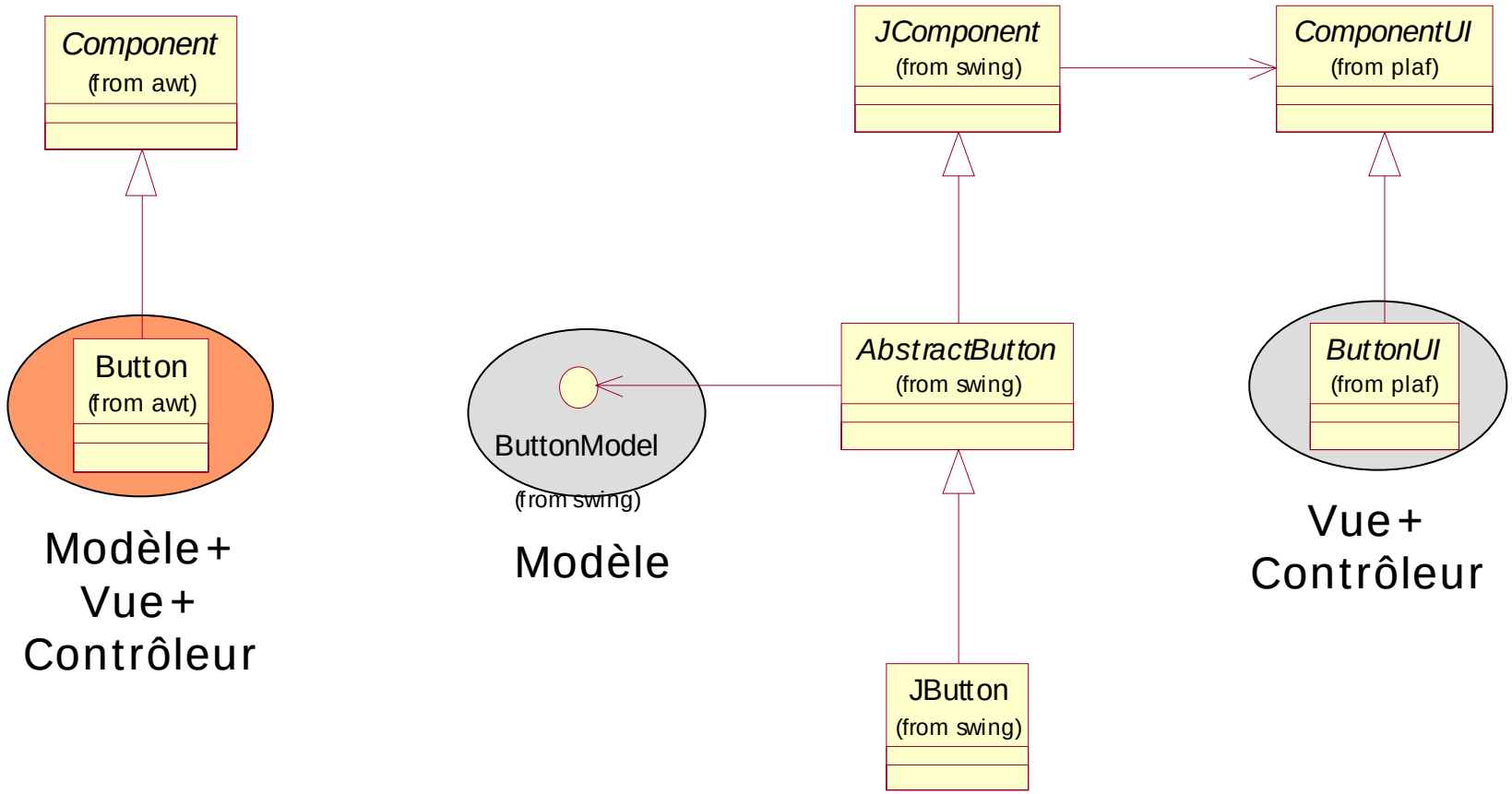
- Architecture MVC
- Look&feel « plugable »
- On peut développer son propre LAF
- Couche dérivée de AWT
- Ajoute de nombreuses fonctionnalités
- Couche entièrement JAVA
- Dégrade les performances

Architecture MVC (principe)

- MVC : Modèle-Vue-Contrôleur
- Un objet d'interface est représenté par 3 classes:
  - La classe **Modèle** : contient les données et leurs méthodes de manipulation
  - La classe **Vue** : se charge de « dessiner » l'objet
  - La classe **Contrôleur** : permet de gérer l'interaction

AWT

SWING


Component
(from awt)
Button
(from awt)
Modèle+
Vue+
Contrôleur
ButtonModel
(from swing)
Modèle
JComponent
(from swing)
AbstractButton
(from swing)
JButton
(from swing)
ComponentUI
(from plaf)
ButtonUI
(from plaf)
Vue+
Contrôleur

Paramétrage du LAF

- Dans $JAVA_HOME/jre/lib/swing.properties

  swing.defaultlaf=*classe look&feel*

- Exemple :

```
swing.defaultlaf=com.birosoft.liquid.LiquidLookAndFeel
```

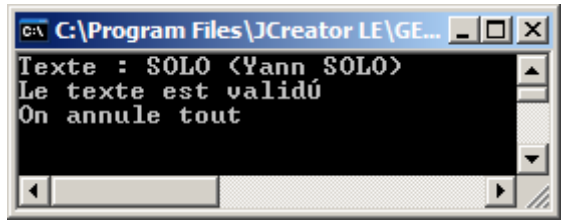
Une petite interface

LAF Windows

LAF Liquid

LAF Metal

Sorties lors de l'exécution



Le développement de l'interface

```
class FenetrePrincipale extends JFrame
{
    public FenetrePrincipale()
    {
        /**
         * Paramétrage de la fenêtre elle-même
         */
        super("La fenêtre principale");
        setSize(400,120);

        /**
         * Création des composants.
         */
        JButton    Ok = new JButton("Ok");
        JButton    Annuler = new JButton("Annuler", new Image
```

Le développement de l'interface

- Trois questions fondamentales :
  - Quels composants et comment ils fonctionnent ?
  - Comment disposer les composants ?
  - Comment gérer les évènements ?

La gestion des évènements

- Gérer un évènement pour un composant
  - Créer une classe écouteur en surchargeant la ou les fonctions appropriées
  - Construire un objet écouteur
  - Associer cet écouteur au composant (éventuellement à plusieurs)

- Classe écouteur = ***Listener***
  - Regroupe un ensemble d'évènements cohérents

La gestion des évènements

- Écouteur = interface de classe JAVA
- Convention de nommage de l'interface :

  `interface XXXListener`

  - WindowListener, etc.

La gestion des évènements

- Évènement = méthode
- Une ou plusieurs méthodes par écouteur
- Convention de nommage :

```
public void nom_evt(XXXEvent)
```

  - Exemple :

```
interface WindowListener {
   public void windowClosing(WindowEvent);
   …
}
```

La gestion des évènements

- Associer un contrôleur à un composant :

```
void addXXXListener(XXXListener)
```

- Un composant peut écouter plusieurs types d'évènements
  - Exemple : JButton peut écouter ActionListener, ChangeListener, ItemListener
- Un contrôleur peut être associé à plusieurs composants

La gestion

- Exemple :

```
class FenetrePrincipale extends JFrame
{
    public FenetrePrincipale()
    {
        // ...

        /**
         * Construction d'un composant
         */
         JTextField T1 = new JTextField(20);

        /**
         * Construction d'un objet contrôleur
         */
         EcouteurTexte ET = new EcouteurTexte();

        /**
         * Association du contrôleur avec le composant
         */
         T1.addActionListener(ET);
    }
}

/**
 * Création de la classe contrôleur
 */
class EcouteurTexte implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
        JTextField  T = (JTextField)(e.getSource());

        System.out.println("Texte : "+T.getText());
    }
}
```

La gestion des évènements

- Rq: toutes les méthodes d'une interface doivent être surchargées
- Pénible si toutes les méthodes ne sont pas utiles

La gestion des évènements

- Exemple :

```
class EcouteurFenetre implements WindowListener
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }

    public void windowActivated(WindowEvent e) { /* ne fait rien */ }
    public void windowClosed(WindowEvent e) { /* ne fait rien */ }
    public void windowDeactivated(WindowEvent e) { /* ne fait rien */ }
    public void windowDeiconified(WindowEvent e) { /* ne fait rien */ }
    public void windowIconified(WindowEvent e) { /* ne fait rien */ }
    public void windowOpened(WindowEvent e) { /* ne fait rien */ }
}
```

La gestion des évènements

- Utilisation des adaptateurs
- Adaptateur = implémentation par défaut d'un écouteur
- Convention de nommage :

  `class XXXAdapter`

- Utiliser un adaptateur = étendre XXXAdapteur
- Pas d'adaptateur si l'interface ne contient qu'une méthode

La gestion des évènements

- Exemple :

```
/**
 * Version de l'écouteur avec adaptateur
 */
class EcouteurFenetre extends WindowAdapter
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
}
```

La gestion des évènements

- import java.awt.event.*
  - Les 15 écouteurs de base
- import javax.swing.event.*
  - Pour les évènements spécifiques SWING
  - 23 écouteurs SWING

Les principaux composants SWING

- Les fenêtres : JFrame, JDialog, JWindow
- Manipulation de texte : JLabel, JTextField, JFormattedTextFieldJTextArea, JEditorPane
- Les boutons : JButton, JCheckBox, JRadioButton
- Les listes : JComboBox, JList
- Les container : JTabbedPane, JSplitPane
- Les menus : JMenuItem, etc.
- Les composants complexes : JTree, JTable
- Les fenêtres de dialogue prédéfinies : JFileChooser, JOptionPane etc.

Les fenêtres

- Toutes les fenêtres
  - Sont créées cachées
  - setVisible(boolean) : montre ou cache la fenêtre
  - dispose() : détruit physiquement la fenêtre
  - setSize(int L, int H) : fixe les dimensions de la fenêtre
  - getContentPane() : renvoie le conteneur qui contiendra les composants fils
  - Écouteur : WindowListener, WindowAdapter

Les fenêtres

- JFrame, JDialog
  - JFrame non modale, JDialog modale
  - JFrame(String titre) : le constructeur standard
  - JDialog(String titre, Frame parent, boolean modalité) ou bien JDialog(String titre, Dialog parent, boolean modalité) : constructeurs standard
  - setJMenuBar(JMenuBar M) : ajoute une barre de menus déroulants
  - JFrame.setDefaultLookAndFeelDecorated(true) : utilise le LAF pour les JFrame
- JWindow
  - Fenêtre sans bordure ni barre d'en-tête

Les fenêtres

- WindowListener:

void **windowActivated**(WindowEvent e) : reçoit le focus
void **windowClosed**(WindowEvent e) : à la destruction
void **windowClosing**(WindowEvent e) : à la fermeture
void **windowDeactivated**(WindowEvent e) : perd le focus
void **windowDeiconified**(WindowEvent e) : réduction
void **windowIconified**(WindowEvent e) : agrandissement
void **windowOpened**(WindowEvent e) : à la création

- WindowEvent

Window **getWindow**() : renvoie la fenêtre qui déclenche l'évènement

Les fenêtres

- Exemple:

Les boutons

- JButton

  JButton(String text)

  JButton(String text, Icon icon)

- ActionListener

  void **actionPerformed**(ActionEvent e)

- ActionEvent

  Object **getSource**()

Les boutons

- JCheckBox

  JCheckBox(String text)

  JCheckBox(String text, boolean selected)

  JCheckBox(String text, Icon icon, boolean selected)

  boolean **isSelected**()

- ItemListener (s'il est dans un groupe)

  void **itemStateChanged**(ItemEvent e)

- ItemEvent

  Object **getItem**() : renvoie le bouton modifié

  int **getStateChange**() : renvoie le nouvel état

Les boutons

- JRadioButton idem JChekBox
- Se place dans un ButtonGroup

- ButtonGroup

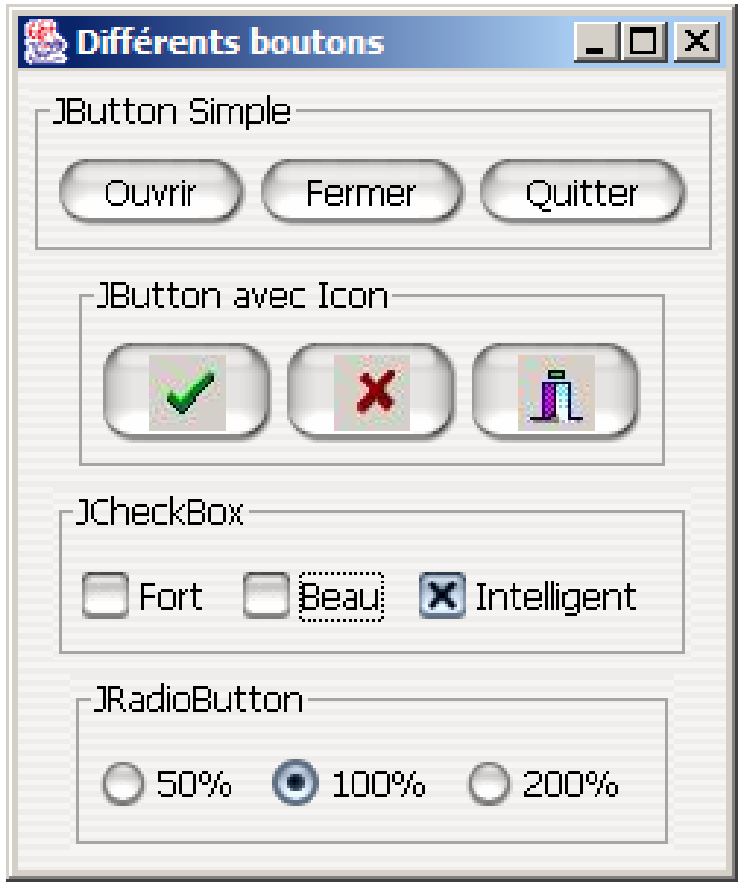
  ButtonGroup()

  void **add**(AbstractButton b)

- AbstractButton classe de base des boutons SWING

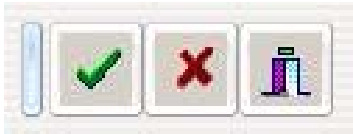
Les boutons

- Exemple :



Les barres de boutons

- JToolBar
  - **JToolBar**(int orientation)
    - HORIZONTAL, VERTICAL
  - Component **add**(Component comp) : pour ajouter un bouton avec icône



Les listes

- JComboBox
- JList

JComboBox

- Affiche une seule option d'une liste défilante

- **JComboBox**(**Object**[] items) : liste initialisée à partir des objets du tableau (toString doit être implémentée)
- **JComboBox**() : liste vide
- void **addItem**(Object anObject) : ajoute une option en fin de liste

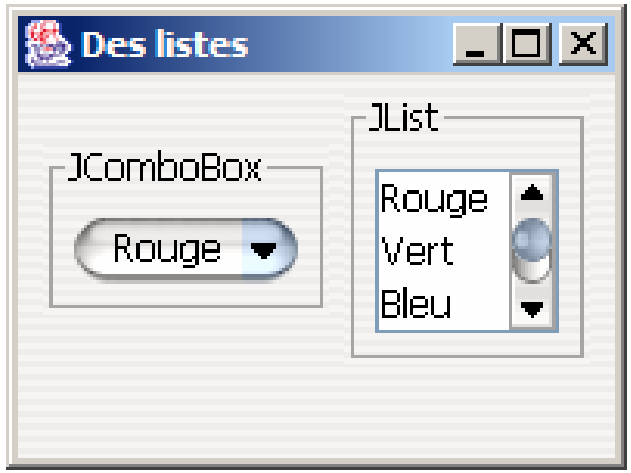
- void **addItemListener**(ItemListener aListener)

JList

- Affiche une liste d'option
- Associé à JScrollPane pour le défilement
- **JList**(Object[] listData) : initialisée à partir des objets du tableau (toString doit être implémentée)
- void **setVisibleRowCount**(int visibleRowCount)
- void **addListSelectionListener**(ListSelectionListener l)

- ListSelectionListener
  - void **valueChanged**(ListSelectionEvent e)
- ListSelectionEvent
  - int **getFirstIndex**()
  - int **getLastIndex**()

Exemple de listes



Les containers

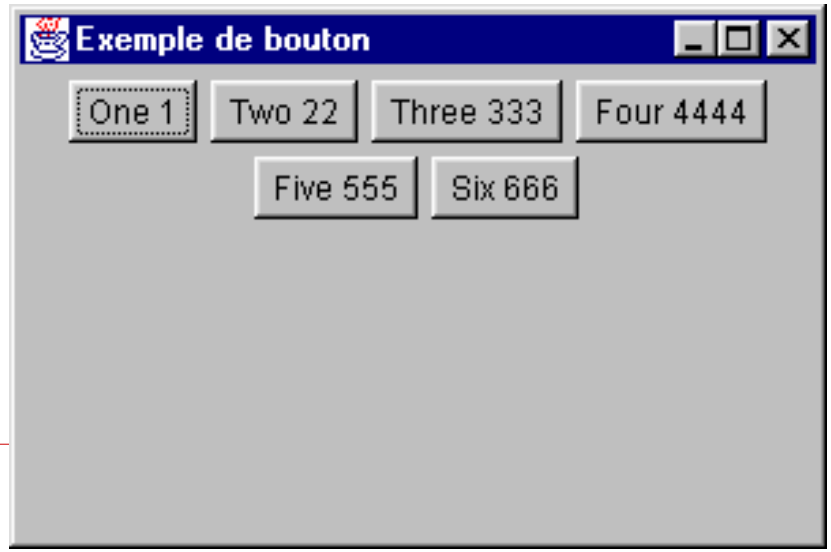
- JPanel
- JTabbedPane
- JSplitPane
- JDesktopPane

JPanel

- JPanel
  - Contient d'autres composants (y compris JPanel)
  - **JPanel**() : construcion par défaut
  - **JPanel**(LayoutManager layout) : construction à partir du gestionnaire de placement
  - void **setLayout**(LayoutManager mgr): gestionnaire de placement des fils
  - Component **add**(Component comp) : ajoute un composant dans le JPanel
- Layout
  - FlowLayout
  - BorderLayout
  - GridLayout
  - BoxLayout

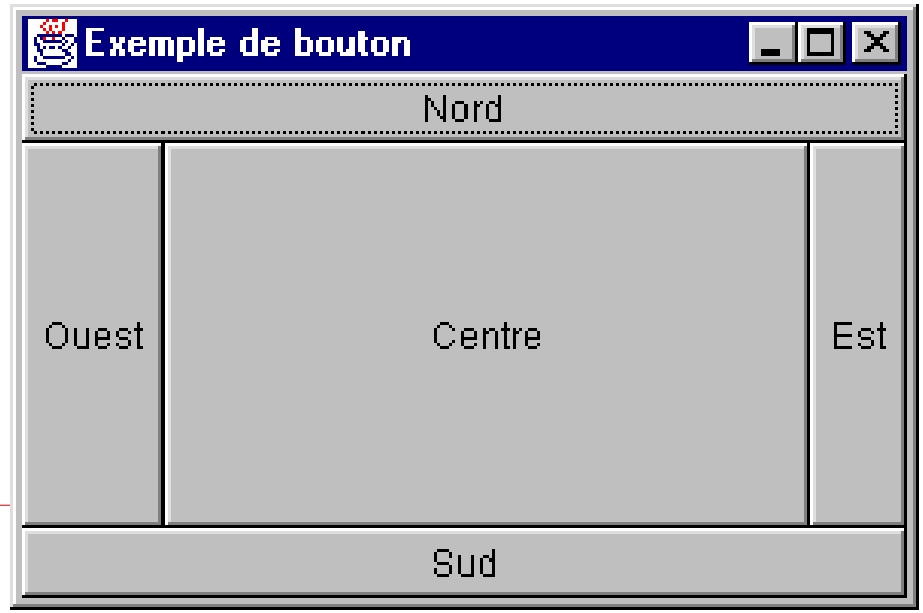
FlowLayout

- Placement des composants de gauche à droite dans l'ordre de construction
- Les composants ont leur juste taille
- Centrage par défaut
- La réduction du JPanel peut provoquer des « retours à la ligne »
- **FlowLayout**()



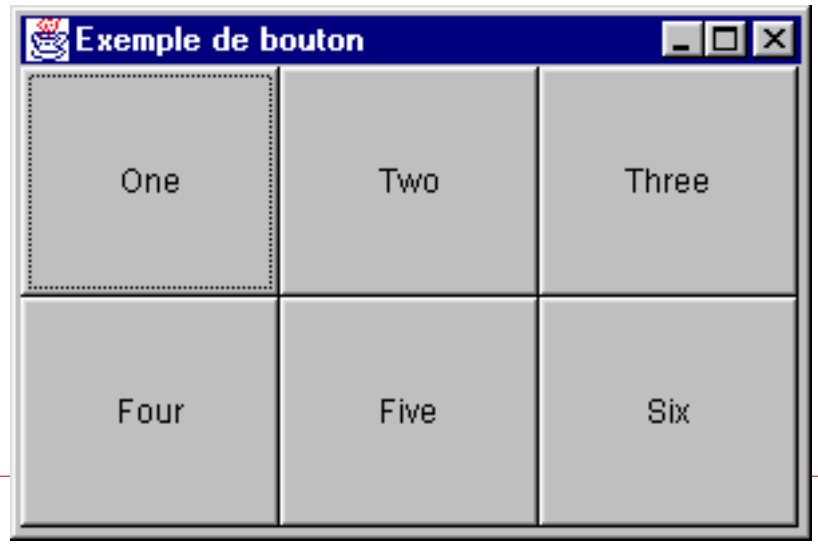
BorderLayout

- La zone est découpée en 5 régions
  - NORTH,SOUTH,WEST,EAST,CENTER
- Le panel ne peut contenir que 5 composants
  - MonPanel.**add**(Comp1,**BorderLayout.NORTH**)
  - Les composants prennent la taille de la région
  - **BorderLayout**()



GridLayout

- Matrice de composants
- Les composants prennent la taille des cellules
- Chaque cellule ne contient qu'un composant
- Les composants sont ajoutés ligne par ligne
- **GridLayout**(int nblig, int nbcol)



BoxLayout

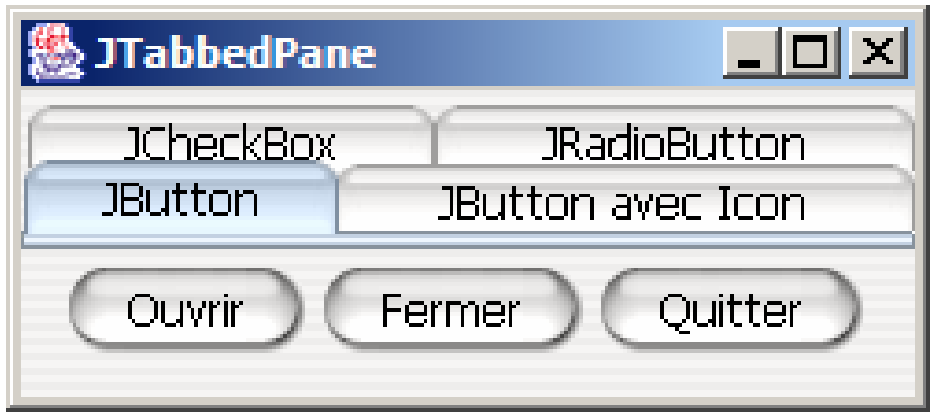
- Idem FlowLayout mais l'alignement est soit horizontal (X_AXIS), soit vertical (Y_AXIS)
- **BoxLayout**(Component C, int Alignement)
- Exemple :

```
MonPanel.setLayout(new
    BoxLayout(MonPanel,BoxLayout.Y_AXIS));
```

- Présentation complexe : imbrication des JPanel

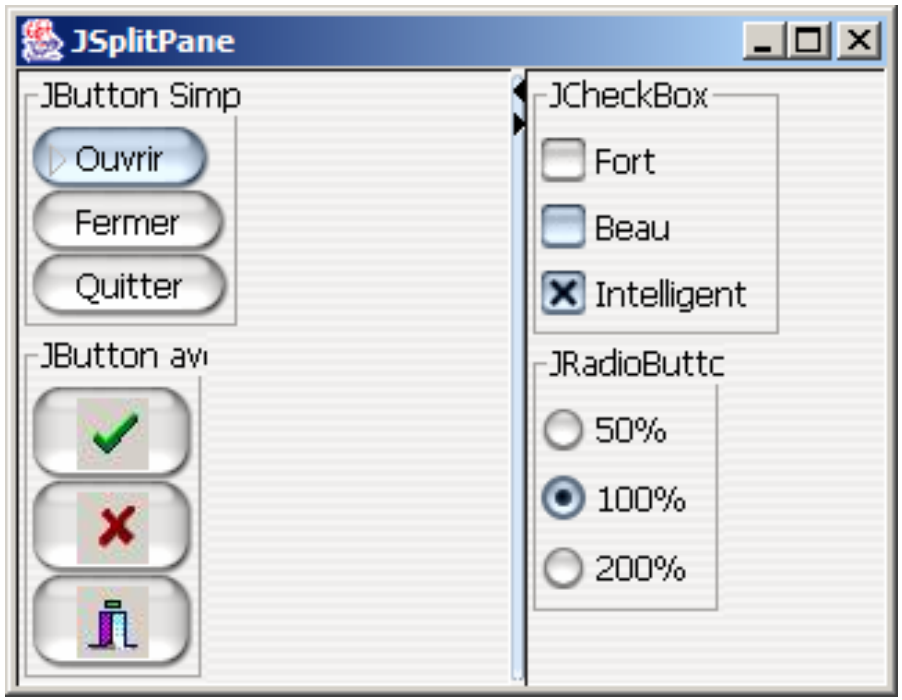
JTabbedPane

- JTabbedPane
  - **JTabbedPane**() : Panneau vide
  - void **addTab**(String titre, Component comp)
  - En général, comp est un JPanel



JSplitPane

- JSplitPane



JSplitPane

- **JSplitPane**(int Orientation,
  Component A,
  Component B)

Si Orientation = JSplitPane.**HORIZONTAL_SPLIT**

A = Gauche et B = Droit

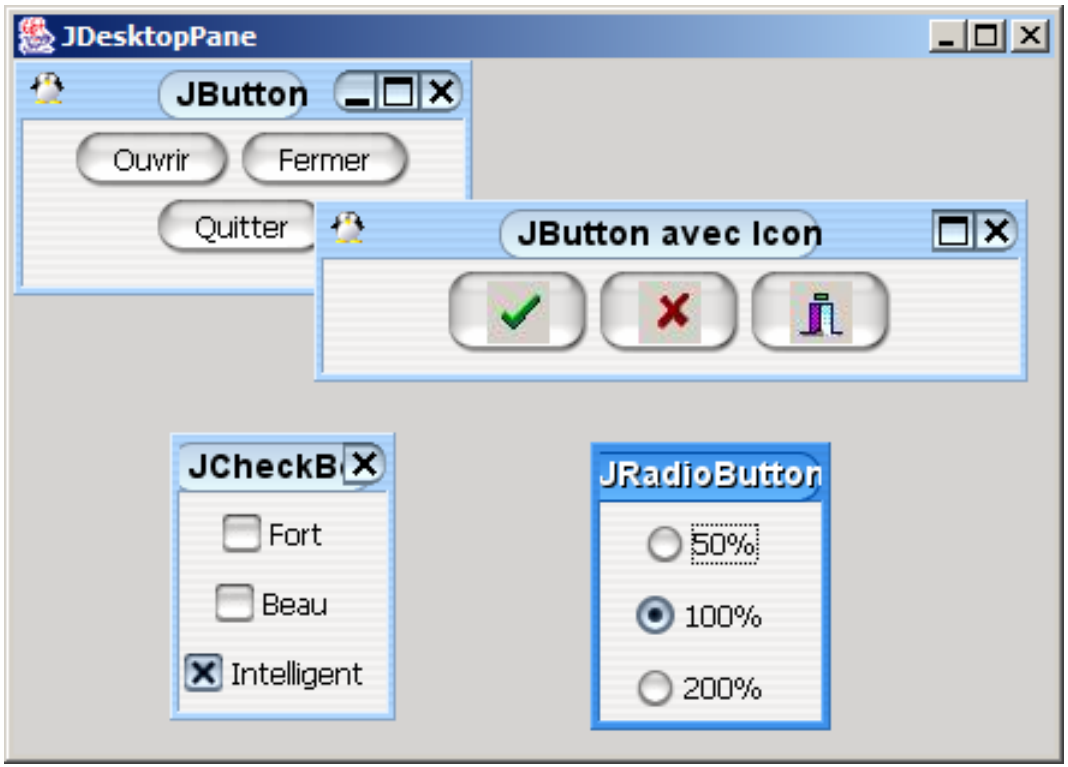
Si Orientation = JSplitPane.**VERTICAL_SPLIT**

A = Haut et B = Bas

- void **setDividerLocation**(int pos) : fixe la position initiale du séparateur (en pixels)
- void **setContinuousLayout**(boolean B) : détermine si l'affichage est continuellement remis à jour lors du déplacement du séparateur
- void **setOneTouchExpandable**(boolean B) : affiche les flèches dans le séparateur

JDesktopPane

- JDesktopPane



JDesktopPane

- JDesktopPane
  - **JDesktopPane**() : panneau vide
  - void **add**(JInternalFrame F)

  - **JInternalFrame**
    - Classe de fenêtre identique à Frame
    - **JInternalFrame**(String titre,
      boolean resizable,
      boolean closable,
      boolean maximizable,
      boolean iconifiable)

Manipulation de texte

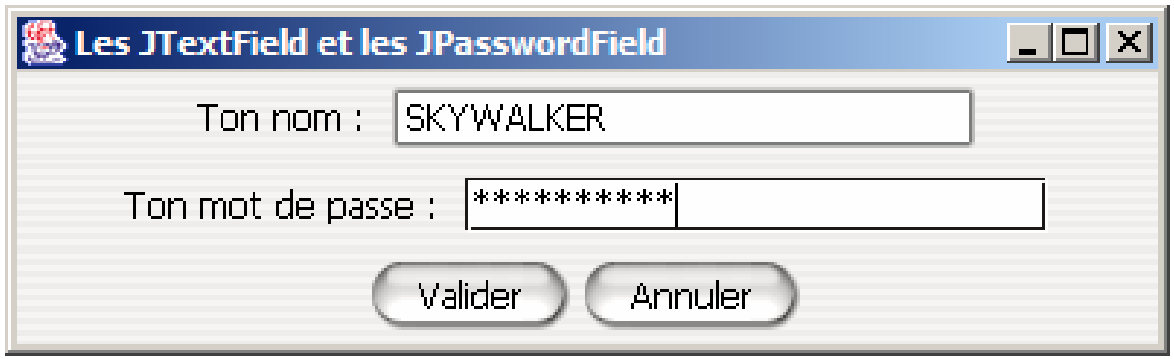
- JTextField, JPasswordField
- JTextArea
- JEditorPane
- (JTextPane, JFormattedTextField)

JTextField, JPasswordField

- JTextField
  - **JTextField**(int columns)
  - String **getText**()
  - void **setText**(String t)
  - void **setEditable**(boolean b)
  - ActionListener : se déclenche par un <RC>
- JPasswordField = JTextField : les caractères saisis sont remplacés par un caractère de substitution (‘*’ par défaut)
  - void **setEchoChar**(char c)

JTextField, JPasswordField

- Exemple :



JTextArea

- JTextArea
  - **JTextArea**()
  - void **append**(String str) : ajoute une chaine dans le JTextArea
  - CaretListener : changement de position du curseur
  - KeyListener : évènements clavier
  - Généralement associé à JScrollPane pour obtenir le défilement du Texte

JTextArea

- JScrollPane
  - **JScrollPane**(Component view) : fait apparaître les barres de défilement selon les dépassements

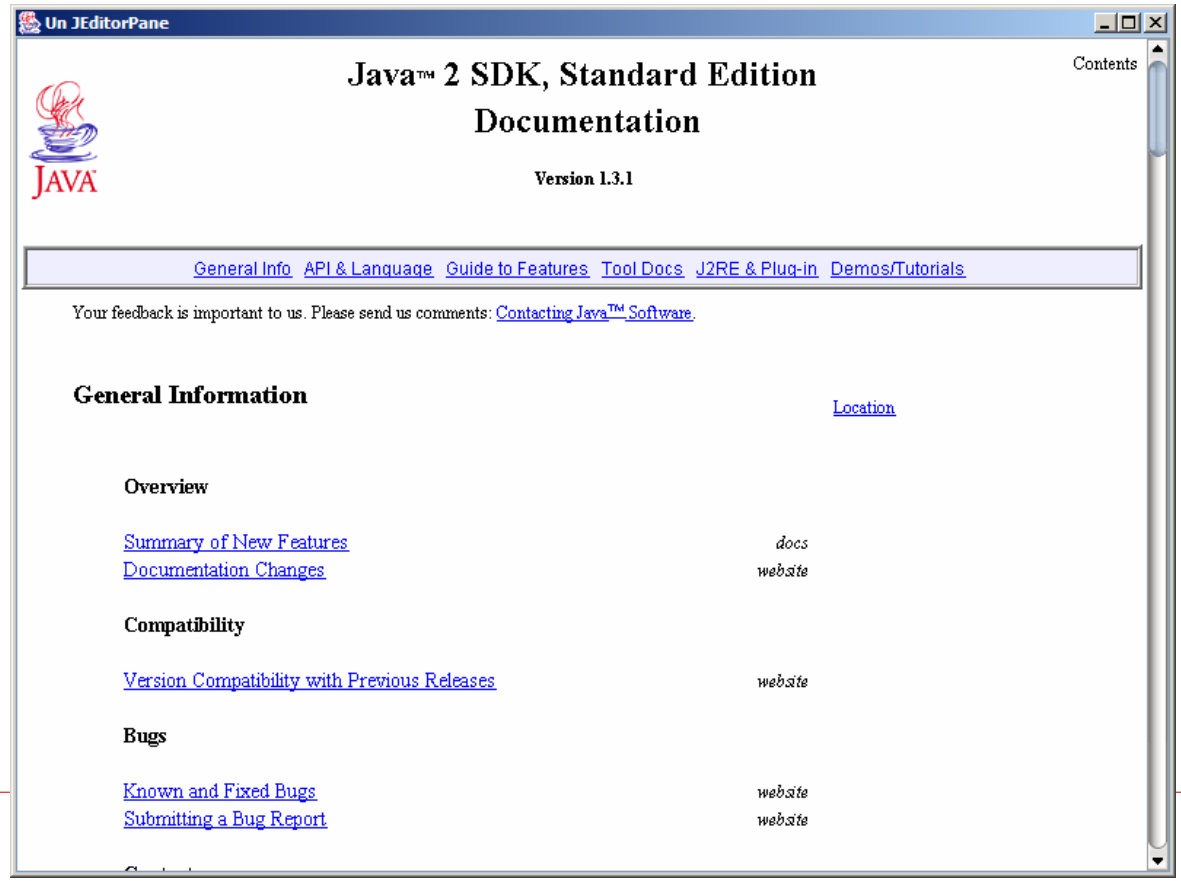
```
/**
 * Le JTextArea associé à un JScollPane
 */
JTextArea   ta = new JTextArea();
JScrollPane sp = new JScrollPane(ta);

ta.append("Bonjour");
```

JEditorPane

- Affiche du texte normal, HTML ou RTF
- **JEditorPane**() : construction
- void **setPage**(String url) ou void **setPage**(URL url) : charge une page HTML
- void **setEditable**(false) : pour que les hyperliens soient sensibles
- HyperlinkListener : écouteur pour les hyperliens
  - HyperlinkEvent.EventType **getEventType**() : ACTIVATED, ENTER, EXITED
  - URL **getURL**()

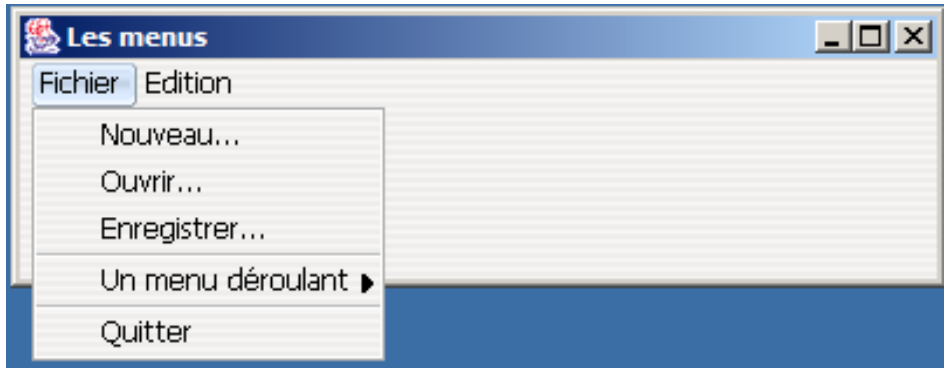
JEditorPane

- Exemple :



La barre des menus

- Propriété des JFrame et des JDialog
- JMenuBar
  - C'est la barre de menu
  - **JMenuBar**()
  - JMenu **add**(JMenu c)
- JMenu
  - Menu déroulant
  - Dérive de JMenuItem
  - **JMenu**(String s)
  - JMenuItem **add**(JMenuItem menuItem)
  - void **addSeparator**()
- JMenuItem
  - Option terminale
  - Voir JButton, ActionListener, etc.
- JCheckBoxMenuItem, JRadioButtonMenuItem
  - Dérivée de JMenuItem
  - Voir JCheckBox, JRadioButton



Les fenêtres de dialogue prédéfinies

- JOptionPane
- JFileChooser
- JColorChooser

JOptionPane

- Fenêtres de message prédéfinie
- **JOptionPane.showMessageDialog**(
  Component Frame,
  Object message,
  String Titre,
  int messageType)

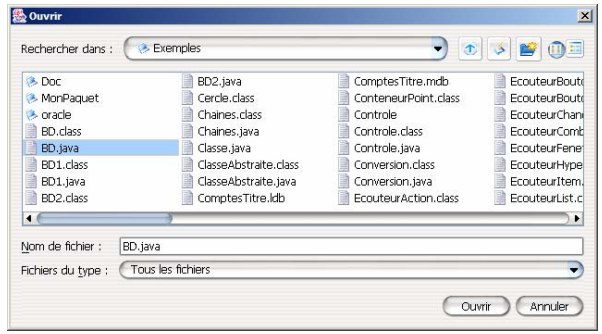
  - messageType : ERROR_MESSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE, QUESTION_MESSAGE, PLAIN_MESSAGE

JOptionPane

- **int JOptionPane.showConfirmDialog(**
  Component Frame,
  Object message,
  String Titre,
  int messageType)

  - messageType : YES_NO_OPTION,etc.
  - résultat : YES_OPTION ou NO_OPTION

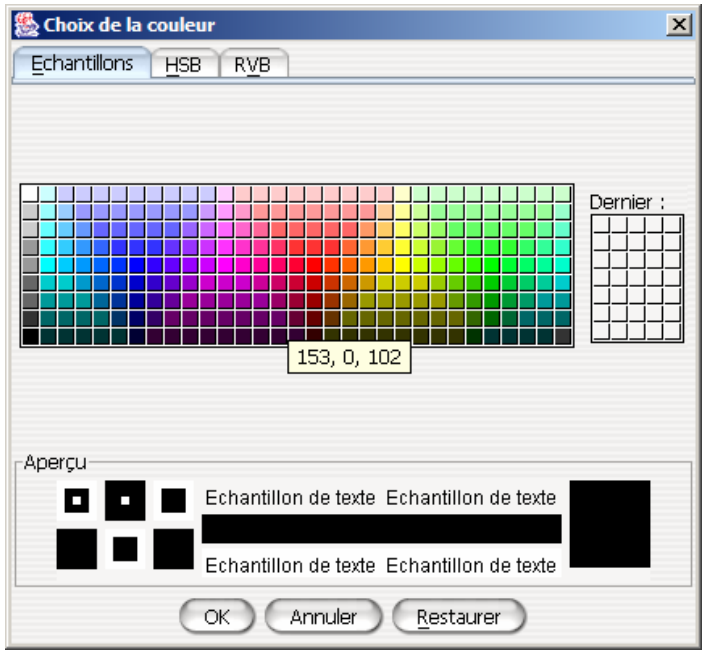
JFileChooser



- **JFileChooser**()
- File **getSelectedFile**()
- File **getCurrentDirectory**()
- int **showOpenDialog**(Component parent)
- int **showSaveDialog**(Component parent)
- Résultat de la fonction
  - JFileChooser.CANCEL_OPTION
  - JFileChooser.APPROVE_OPTION

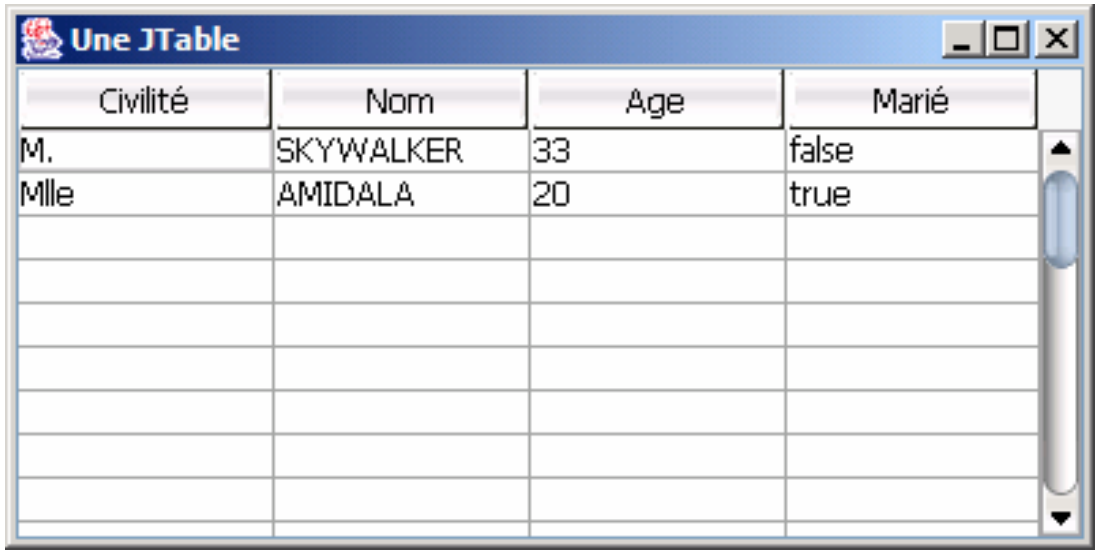
JColorChooser

- static Color **showDialog**(Component comp, String titre, Color CoulInit)



Les composants complexes

- JTable



JTable

- Une table simple
  - **JTable**(Object[][] Données, Object[] Noms) : initialise le tableau avec les dimensions de « Données ». Une ligne spéciale (inaccessible) est ajoutée avec les noms des colonnes
  - Object **getValueAt**(int l, int c) : no comment
  - void **setValueAt**(Object v, int l, int c) : no comment

  - Est généralement placé dans un ScrollPane. Sinon, pas d'en-tête.
  - Tous les éléments d'une colonne doivent être du même type

JTable

- Personnalisation d'une colonne
  - TableColumnModel **getColumnModel**() renvoie toutes les informations sur toutes les colonnes d'une JTable
- TableColumnModel
  - TableColumn **getColumn**(int i) renvoie les informations sur la ième colonne
- TableColumn
  - void **setWidth**(int l)
  - void **setHeaderRenderer**(TableCellRenderer hr)
  - void **setCellRenderer**(TableCellRenderer cr)
  - void **setCellEditor**(TableCellEditor ce)

JTable

- TableCellRenderer
  - Interface contenant une seule méthode abstraite:

```
Component getTableCellRendererComponent(
                    JTable table,
                    Object value, // attention à null
                    boolean isSelected,
                    boolean hasFocus,
                    int row,
                    int column)
```

  - La méthode doit créer un composant modèle à partir des données passées en paramètres

JTable

- Exemple:

```
class AffichageMarie extends JCheckBox implements TableCellRenderer
{
    public Component getTableCellRendererComponent(JTable table,
                                                   Object value,
                                                   boolean isSelected,
                                                   boolean hasFocus,
                                                   int row,
                                                   int column)
    {
        setSelected(false);
        if (value != null)
        {
            Boolean b = (Boolean)value;
            setSelected(b.booleanValue());
        }
        return this;
    }
}
```

Paramétrage de la JTable

```
t.getColumnModel().getColumn(3).setCellRenderer(new AffichageMarie());
```

JTable

- TableCellEditor
  - Interface contenant plusieurs méthodes abstraites dont les principales sont:

Component **getTableCellEditorComponent**(
JTable table,
Object value,
boolean isSelected,
int row,
int column)

  - La méthode doit créer un composant modèle à partir des paramètres, et qui se chargera de l'acquisition de l'information

JTable

Object **getCellEditorValue**()

- Renvoie l'objet contenant la valeur saisie (Integer pour un int, Boolean pour un boolean, etc.)
- Pour éviter d'implémenter les autres, il faut étendre AbstractCellEditor

JTable

- Exemple :

```
class EditionMarie extends AbstractCellEditor implements TableCellEditor
{
    private JCheckBox   cb;

    public EditionMarie()
    {
        cb = new JCheckBox();
    }

    public Object getCellEditorValue()
    {
        return new Boolean(cb.isSelected());
    }

    public Component getTableCellEditorComponent(JTable table,
                                                 Object value,
                                                 boolean isSelected,
                                                 int row,
                                                 int column)
    {
        return cb;
    }
}
```

JTable

- Il est également possible:
  - De définir des startégies de sélection (ListSelectionModel)
  - De définir le comportement « fin » d'une JTable (TableModel)

Autres composants

- JSlider

- JProgressBa

- JSpinner

Graphisme

- Graphics
  - Ressource contenue dans *Component*
    - Graphics getGraphics()
  - Contient la plupart des fonctions graphiques

```
void drawArc(int xhg, int yhg, int lbe, int hbe, int startAngle, int arcAngle)
void drawLine(int x1, int y1, int x2, int y2)
void drawRect(int x, int y, int width, int height)
void drawString(String str, int x, int y)
void fillRect(int x, int y, int width, int height)
void setColor(Color c)
...
```

Graphisme

- L'objet *Graphics* n'est valide qu'après le premier événement *paint*
- Un Container ou un composant Canvas capte l'événement paint en surchargeant la fonction : *void paint(Graphics)*
- Exemple:

```
public class MaFenetre extends Frame
{
  public MaFenetre()
  {...}

  public void paint(Graphics g)
  {
    g.drawLine(0,0,10,10);
  }

  ...
}
```

Graphisme

- Attention au recouvrement
  - Exemple : soit une *Frame* contenant un *Panel*

```
public class MaFenetre extends Frame
{

  public MaFenetre()
  {
    ...
    Panel P = new Panel();
    add(P);
    ...
  }

  public void paint(Graphics g)
  {
    g.drawLine(0,0,10,10);
  }
```

La fenêtre ne contient qu'un seul composant

Un segment est tracé dans la fenêtre

Le segment est masqué par le Panel

Graphisme

- La solution :

```
public class MonPanel extends Panel
{
  public void paint(Graphics g)
  {
    g.drawLine(0,0,10,10);
  }
```

Nouvelle classe Panel qui gère le *paint*

```
public class MaFenetre extends Frame
{
  public MaFenetre()
  {
    ...
    Panel P = new MonPanel();
    add(P);
    ...
  }
```

La fenêtre contient le nouveau Panel, et ne gère plus le *paint*